

تقرأ في العدد السابع

- مغامرات في دلفي
- عدة إصدارات من مشروع واحد في دلفي / لازاروس
- تعرف على دوال النظام GetWindowLong / SetWindowLong
- المتباينات أو المتقلبات (Variants) في دلفي
- عرض البيانات تلقائيا في قاعدة Interbase/FireBird باستعمال الأحداث Events
- لمحات لطريق البرمجة
- خواطر برمجية : فكر من جديد

دلفي و ملفات قواعد بيانات MS SQL SERVER (الإتصال قبل/بعد التحزيم و التوزيع)

فهـــرس العدد

✓ مغامرات في دلفي
✓ دلفي و ملفات قواعد بيانات MS SQL SERVER (الإتصال قبل/بعد التحزيم و التوزيع)
✓ عدة إصدارات من مشروع واحد في دلفي / لازاروس
✓ تعرف على دوال النظام GetWindowLong / SetWindowLong
✓ المتباينات أو المتقلبات (Variants) في دلفي
✓ عرض البيانات تلقائيا في قاعدة Interbase/FireBird باستعمال الأحداث Events
✓ لمحات لطريق البرمجة
✓ خواطر برمجية : فكر من جديد

DELPHI4ARAB

مغامرات في دلفي - بقلم خالد الشقروني

يكثر في برامجي استخدامي لدوال تغيير البيانات من نوع لآخر، و خاصة دالتي IntToStr و StrToInt ، ومع كثرة استخدامها و تنوع البيانات المراد تحويلها، تصير الأمور مزعجة بالنظر إلى طول أسماء هذه الأدوات وحرصي على كتابتها بالشكل الملائم بمراعاة الأحرف العالية والمنخفضة.

صحيح أن أصابعي أمست معتادة على طباعة أحرف بعض الإجرائيات و بسرعة كبيرة، لكن يظل الأمر مزعجا خاصة في الإجرائيات التي تتطلب أكثر من مُعطى واحد Arguments أو التي لديها أكثر من توأم overload، وما يتبعها من مراجعة المساعدة للبحث عن الإسم الصحيح للإجرائية المناسبة. ويزداد الأمر ارباكا إذا استخدمت ثلاث أو أربع إجرائيات في تعليمة واحدة، فتجتاز التعليمة الواحدة الفاصل العمودي على يمين المحرر والمحدد بثمانين حرفا مما يؤثر على مقروئية البرنامج و تتبع خطواته (أو هكذا يقولون).

لذا فكرت بأن أجد طريقة أريح بها أصابعي ودماغي من عناء تذكر وكتابة الأسماء الطويلة لإجرائيات ودوالّ تحويل نوع البيانات، وذلك بتغليفها في إجرائيات ذات أسماء أقصر، وتوحيدالمتشابه منها في إسم واحد ما أمكن ذلك مستغلا ميزة إعادة التحميل overload عند تعريف الإجرائيات.

بدأت بموضوع تحويل الأرقام إلى حروف. و بالذات مع الدالة IntToStr التي تقوم بتحويل رقم ذو عدد صحيح إلى أحرف نصية، مثل التالي:

```
Caption := IntToStr(i);
```

هذه الدالة تتكون من ثمانية أحرف، فقمت بتغليفها wrap داخل دالة function يكون إسمها أقصر، فاخترت الاسم "S_" بحيث تكون كالتالي:

```
function _S(const X: integer): string;
begin
  result := IntToStr(X);
end;
```

بذلك كلما أردت تحويل عدد صحيح إلى نصّ أستدعي هذه الدالة:

```
Caption := _S(i);
```

لكن هذه فقط تقوم بتحويل العدد الصحيح Integer ماذا لو كان العدد من نوع Single ؟

لحسن الحظ كل هذه الأنواع يمكن جمعها تحت النوع Extended وإنشاء دالة لتحويل هذا النوع وتسمية هذه الدالة بنفس اسم الدالة الأولى أي S_ مع تزيينها بأمر overload . أولا قمت بإضافة الأمر overload للدالة السابقة:

```
function _S(const i: integer): string; overload;
```

ثم عرّفت دالة أخرى مع محدد parameter من نوع extended :

```
function _S(const AValue: Extended): string; overload;
```

و هذا جسم الدالة:

```
function _S(const AValue: Extended): string;
begin
  result := FloatToStr(AValue);
end;
```

لم لا نضع مزيدا من السكّر ونصنع دالة أخرى نجعلها تقوم بضبط عدد الخانات بعد الفاصلة:

```
function _S(const AValue: Extended; Digits: Integer): string;
overload;
begin
  result := FloatToStrF(AValue, ffFixed, 16, Digits);
end;
```

بهذا يمكن تطبيق هذه الدالة على الرقم 45687.245654 بحيث تعطينا تمثيل نصي بثلاث خانات بعد الفاصلة

```
_S(45687.245654, 3)
```

فنتحصّل على 45687.246.

من بين الأنواع الأخرى التي احتجت لتسهيل عملية تحويلها إلى نص؛ النوع OleVariant، والتي كثر استخدامي لها عند التعامل المباشر مع قيم الحقول في قواعد البيانات تحت مظلة ADO. أيضا كانت عونا كبيرا لي لاختصار الوقت ومجهود الكتابة عند التعامل مع قيم عناصر XML:

```
function _S(const Value: OleVariant): string;
begin
  if Value <> null then
    result := Value
  else
    result := '';
end;
```

التحويل إلى نص بنفس مسمّى الدالة يمكن أن يشمل التركيبات records ، مثلا هذه الدالة تقوم بتحويل قيمة من نوع TPoint إلى نص:

```
function _S(const APoint: TPoint): string;
begin
  Result := Format('(%d,%d)', [APoint.X, APoint.y]);
end;
```

لأحصل على تمثيل نصي لقيمة من نوع TPoint مثل : (400,600)

و أخرى لمصفوفة من عناصر نوع TPoint:

```
function _S(const Points: array of TPoint): string; overload;
var
  i: integer;
begin
  Result := '';
  for i := 0 to high(Points) do
  begin
    Result := result + Format('(%d,%d) ',[Points[i].X, Points[i].y]);
  end;
  Result := Trim(result);
end;
```

أيضا أخرى خاصة بالنوع TRect

```
function _S(const ARect: TRect): string;
begin
  Result := Format('(%d,%d)',[ARect.TopLeft.X, ARect.TopLeft.Y]) + ' '
      + Format('(%d,%d)',[ARect.BottomRight.X, ARect.BottomRight.Y]);
end;
```

هذه التحويلات تسهّل العمل كثيرا عندما يتعلق الأمر بتسجيل الكيانات التي تحوي قيم من هذه الأنواع في ملفات نصية مثل ملفات log أو عند عمليات serialization والتحويل من كائنات إلى صيغ أخرى مثل XML .

تحويل نوع التاريخ

للأسف لم أستطع صنع دالة لتحويل نوع التاريخ إلى نصّ تحت نفس الإسم أي: S_ ، وذلك لأن نوع التاريخ TdateTime هو في الأصل من نوع Double ، ولهذا إذا مررّنا للدالة قيمة من نوع TDateTime فربما يتعامل معها كرقم وتعطينا التمثيل النصي لهذا الرقم، كذلك إذا مرّرنا قيمة من نوع Double فربما تأتينا النتيجة على صيغة نصية للتاريخ والوقت. الكود التالي يوضح هذا الإشكال:

```
var
  Adouble: double;
begin
  Adouble := 123456.123456;

  caption := _S(Now)  + ' * ' +
             _S(123456.123456) + ' * ' +
             _S(Adouble);
```

الناتج سيكون

```
'2016-03-19 01:02:01 * 123456.123456 * 2238-01-03 02:57:46'
```

لاحظ أن الناتج يختلف بين القيمة التي مرّرت مباشرة للدالة و تلك التي تمّ تمريرها كمتغيّر من نوع double يحمل نفس القيمة.

لذلك استحدث دالة بإسم آخر : _SD لتحويل التاريخ إلى نص:

```
function _SD(const ADateTime: TDateTime): string;
begin
  result := DateTimeToStr(ADateTime);
end;
```

هذه بعض ما يمكن اختصاره من إجرائيات ودوالّ التحويل من أنواع مختلفة إلى نص.

بنفس السياق قمت باستحداث دوال لتحويل نوع نصي string إلى نوع integer بتسمية _i مثل:

```
function _I(const S: string): integer;
begin
  result := StrToIntDef(S, 0);
end;
```

و أخرى لتحويل الأرقام العشرية إلى عدد صحيح:

```
function _I(const AValue: real): integer;
begin
  result := Round(AValue);
end;
```

كذلك تحويل الصيغ النصية للتاريخ إلى نوع تاريخ، بدالة ذات اسم مختصر:

```
function _D(S: string): TDateTime;
begin
  result := StrToDateTime(s);
end;
```

من الأسماء إلى الأفعال

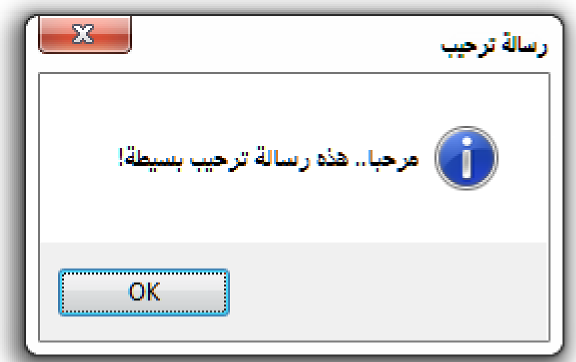
ما قمت به لحدّ الآن هو إنشاء أسماء موجزة لدوال ذات أسماء طويلة.

ماذا عن المهام ذات التعليمات الطويلة، والتي يتكرّر استخدامي لها؟ ماذا لو قمت بتغليف تعليمات هذه المهام داخل دوال مختصة، بحيث كلما احتجت لتنفيذ مهمة منها أكتفي فقط باستخدام الدالة الخاصة بها دون الخوض في تفاصيل تعليماتها.

لنجرّب:

مربعات حوار الرسائل Message dialog box

إذا كنت مثلي مما يفضلون استخدام مربعات حوار الرسائل الأصلية الخاصة بويندوز؛ فحتما تعلم مدى تعدد وتنوع المعطيات والخيارات اللازم إعدادها لإنشاء مربع رسالة بسيطة مثل هذه:



لإنشاء هذه الرسالة يتطلّب الأمر الكود التالي:

```
sMsg := 'مرحبا .. هذه رسالة ترحيب بسيطة!';
sTitle := 'رسالة ترحيب';
result := MessageBoxW(Application.Handle, PWideChar(sMsg),
                        PWideChar(sTitle),
                        MB_ICONINFORMATION +
                        MB_Right +
                        MB_RTLREADING);
```

ويزداد الأمر إرباكا إذا أرت أنواعا أخرى من الرسائل للتنبيه أو الخطأ، أو الطلب من المستخدم الاختيار بين عدة خيارات بالموافقة أو الإلغاء أو تجاوز الأمر، فكل هذه الخيارات عليك أن تصيغها ضمن أمر واحد وهو: MessageBox . لذا وجدت أن الأمر سوف يكون أكثر سهولة لو قسمت أنواع الرسائل بحيث يكون لكل نوع دالّة خاصة به، مثلا واحدة لرسالة استعلام بنعم أو لا، وأخرى تنبيه بموافق و إلغاء الأمر، وهكذا.

بدأت بدالّة عامة أسميتها Msg مشابهة لدالة MessageBox وبنفس محدداتها عدا أن محدّدا نصّ الرسالة والعنوان جعلتهم من نوع string بدل PChar لتسهيل مخاطبتها. هذه الدالة العامة سوف أستخدمها لكي يتم مناداتها من داخل الدوال الأخرى التي اعتزم إنشاؤها.

```
function Msg(Handle: integer; sMsg, sTitle: Widestring;
                                          iType: integer): integer;
begin
  result := MessageBoxW(Handle, PWideChar(sMsg), PWideChar(sTitle),
                                                      iType);
end;
```

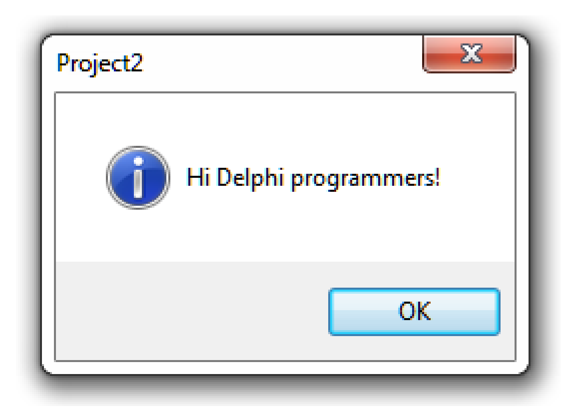
ثم بدأت بإنشاء دالة لعرض نص رسالة مع أيقونة information وزرّ موافق كالتالي:

```
function Msg(sMsg: WideString): integer; overload;
begin
  result := Msg(Application.Handle, sMsg, Application.Title,
                                              MB_ICONINFORMATION);
end;
```

بهذه الدالة يمكنني الآن أن أنشئ مربع رسالة بسيط بأقل ما يمكن من كود مثل التالي:

```
Msg('Hi Delphi programmers!');
```

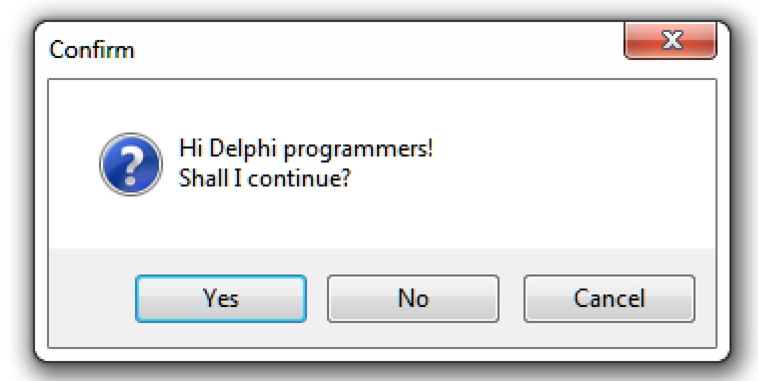
لأتحصّل عل مربع الرسالة التالية:



و هذه :

```
function MsgYesNoCancel(sMsg, sTitle: WideString): integer;
begin
  result := Msg(Application.Handle, sMsg, sTitle, MB_YESNOCANCEL
                                                 + MB_ICONQUESTION);
end;
```

```
MsgYesNoCancel('Hi Delphi programmers!'+ #13 + 'Shall I continue?',
               'Confirm');
```

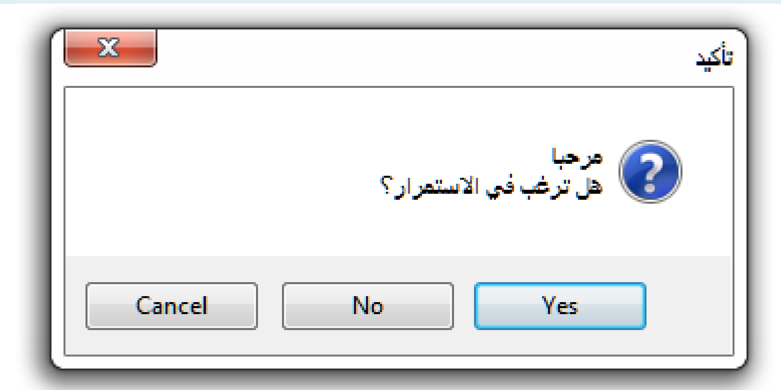


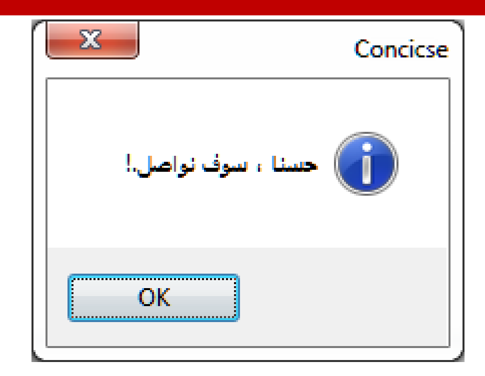
كل دالة لها أخت لها تشبهها لكنها معدّة للعرض من اليمين لليسار لتكون مناسبة للرسائل باللغة العربية.

```
function MsgYesNoCancelR(sMsg, sTitle: WideString): integer;
begin
  result :=  Msg(Application.Handle, sMsg, sTitle, MB_YESNOCANCEL +
                                                  MB_ICONQUESTION +
                                                  MB_RIGHT +
                                                  MB_RTLREADING);
end;
```

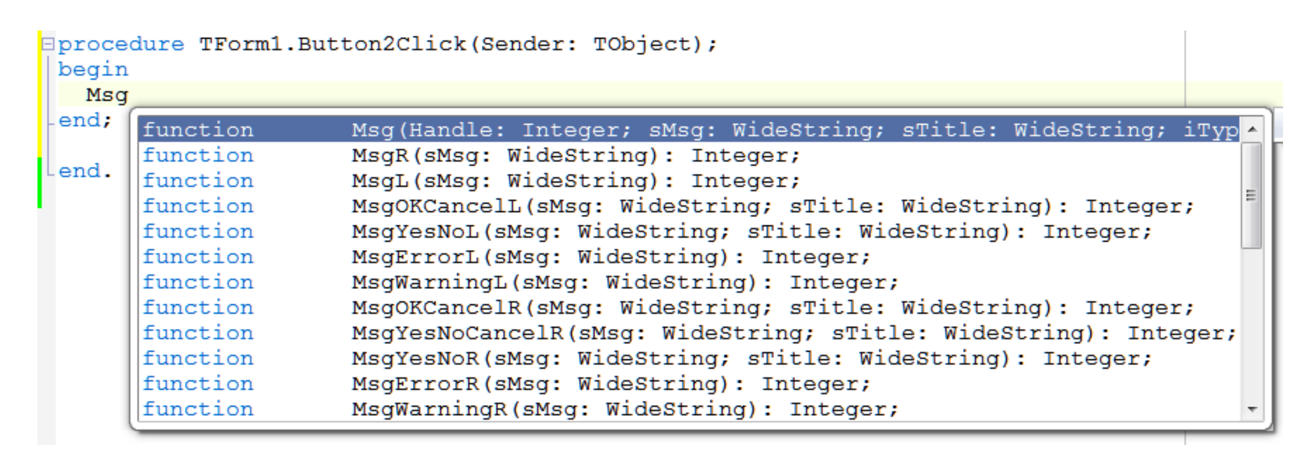
و هذا مثال على تنفيذها:

```
if MsgYesNoCancelR('مرحبا'+ #13 + 'هل ترغب في الاستمرار؟' , 'تأكيد') =
          ID_YES then
  Msg('I will do it!');
```



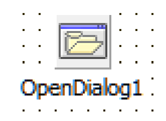


برغم طول أسماء هذه الدوال حتى يسهل الاستدلال على مهامها؛ إلا أن تذكّر أسماء هذه الدوال لن يكون مشكلة؛ حيث أن كلّ دالة تبدأ ب Msg وبمجرد كتابتها فإن قائمة الاستشعار الذكي للكود intellisense سوف تظهر كافة الدوال التي تبدأ بنفس الأحرف للإختيار من بينها.



المزيد من مربعات الحوار

جانب آخر من مربعات الحوار التي أحتاجها دائما هي مربع حوار فتح ملف TOpenDialog و حفظ ملف TSaveDialog ، هذا النوع من مربعات الحوار متوفرة في دلفي كمكوّن مرئي مثل TOpenDialog:

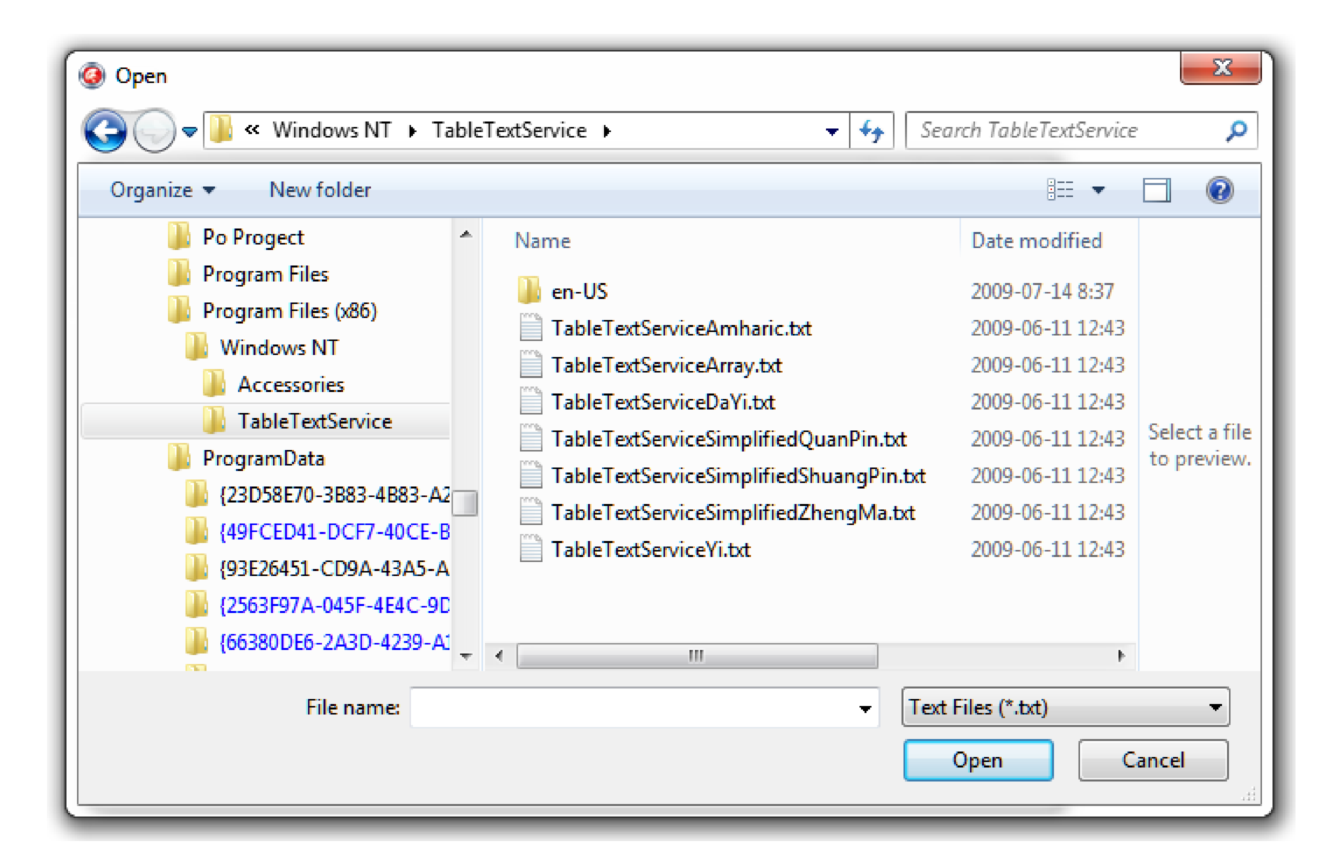


تثبيت هذا المكون بطريقة مرئية قد يكون مزعجا إذا استخدمته في أكثر من نموذج، أو في وحدات كود ليست مرتبطة بواجهة رسومية، من ناحية أخرى، إذا تعاملت معه بدون زرعه كمكوّن مرئي ، فإن ذلك يتطلّب أسطرا متعددة من التعليمات البرمجية مما يشوش على منطقية تسلسل برنامجي بحشوه بما ليس له علاقة.

لذلك قمت بتغليف تعليمات إنشاء هذا المكوّن في دالة خاصة أسميتها DialogOpenFile، بحيث أتحصّل على إسم ملف من المستخدم كالتالي:

```
sFileName := DialogOpenFile('');
```

كي أحصل على هذه النتيجة:



تستقبل الدالة DialogOpenFile معطى argument نصي واحد يتم فيه تحديد نوع الملفات التي يتم فرزها عن غيرها،

مثل:

```
sFileName := DialogOpenFile('Doc Files|*.doc; *.docx|All Ailes|*.*');
```

في حالة عدم تحديد نوع الملفات كما في مثالنا الأول؛ تقوم الدالة بافتراض أن الملفات المفروزة ستكون بامتداد txt. فيما يلي كود الدالة:

```
function DialogOpenFile(Filter: string): string;
var
  dlg: TOpenDialog;
begin
  dlg := TOpenDialog.Create(nil);
  try
    if Trim(Filter) = '' then
      Filter := 'Text Files|*.txt|All Ailes|*.*';

    dlg.Filter := Filter;
    if dlg.Execute then
      result := dlg.FileName
    else
      result := '';
  finally
    dlg.Free;
  end;
end;
```

وبنفس التعليمات تقريبا؛ يمكن إنشاء دالّة أخرى خاصة بمربع حوار حفظ ملف مع استبدال مكون TSaveDialog ب TOpenDialog .

الملفات النصية

المجال التالي التي سأتعامل معه في سعيي لتسهيل المهام البرمجية المتكرّرة هي الملفات النصية.

توجد أكثر من طريقة للحصول على محتوى ملف نصي، لكن ربما أسهلها هو استخدام صنفية TStringList class . لكن التعامل معها يتطلب بعض الخطوات من إنشاء لكائن object من نوع هذه الصنفية، ثم استخدامه لتحميل الملف النصي، وأخيرا تحرير الذاكرة من الكائن وإفنائه. الكود التالي يوضح ذلك في أبسط صورة، وبأقل ما يمكن من خطوات:

```
***
var
  st: TStringList;
  FileName: string;
  AText: string;
begin
  FileName := 'Test.txt';
  st := TStringList.Create;
  try
    st.LoadFromFile(FileName, TEncoding.UTF8);
    AText := st.Text;
  finally
    st.Free;
  end;
  ***
  ***
```

كما نرى خطوات كثيرة لمجرد الحصول على المحتوى النصي للملف.

ما أرغب فيه هو تعليمة بسيطة تغنيني عما سبق مثل التالي:

```
var
  AText: string;
begin
  AText := FileText('Test.txt');
  ***
```

حيث يكفي استدعاء دالة واحدة مع إعطائها إسم الملف ، فتقوم الدالة بكل مايلزم وترد لي بالمقابل النص الذي بالملف في متغير.

قبل الشروع في صنع هذه الدالة؛ أحتاج لصنع دالة مساعدة تكون الأساس لصنع دالة FileText ودوال أخرى تقدّم لي محتوى النص بصيغ أخرى. الدالّة المساعدة إسمها : FileStrings

```
function FileStrings(const FileName: string): TStrings;
var
  sl: TStringList;
begin
  result := nil;
  if not FileExists(FileName) then exit;

  sl := TStringList.Create;
  try
    sl.LoadFromFile(FileName, TEncoding.UTF8);
    result := sl;
  except
    sl.Free;
    sl := nil;
  end;
end;
```

الدالّة تستقبل إسم ملف المطلوب فتحه، وناتجها result هو كائن من نوع TStrings . بعد أن تقوم الدالة بالتأكد من وجود الملف، تقوم بإنشاء كائن من نوع مشتق وهو TStringList ليقوم بفتح وتحميل الملف. إذا نجحت العملية يتم إسناد هذا الكائن إلى ناتج الدالّة، في حالة حدوث أي خطأ يتم إفناء الكائن وإعادة قيمته إلى لا شيئ nil.

(لاحظ إننا لم نقم بإفناء الكائن في حالة نجاح عملية فتح الملف، وتركنا مسؤولية هذا الأمر لمن يقوم بإستدعاء هذه الدالة). بعد هذا نشرع في بناء الدالة FileText كالتالي:

```
function FileText(const FileName: string): string;
var
  Strings: TStrings;
begin
  result := '';
  try
    Strings := FileStrings(FileName);
    if Strings <> nil then
       result := Strings.Text;
  finally
    if Strings <> nil then
      Strings.Free;
  end;
end;
```

كما نرى؛ تقوم الدالة بتطبيق الدالة المساعدة FileStrings على إسم ملف، فتتحصّل على قائمة نصية من نوع TStrings بمحتويات الملف (قائمة بأسطر المحتوى النصّي للملف) ، ثم باستخدام خاصية Text يتم اسناد كامل المحتوى في متغير واحد result من نوع string الذي هو ناتج الدالة.

(لاحظ كيف أن هذه الدالة مسؤولة عن إفناء الكائن Strings المتحصّل من دالة FileStrings)

بهذا تكون الدالة قد سهّلت علينا الحصول على محتويات ملف نصي ووضعه في متغير من نوع string .

```
AText := FileText('Test.txt');
Memo1.Text := AText;
```

ماذا لو أردنا الحصول على محتوى نفس الملف في مصفوفة بحيث تنتظم أسطر نصّ الملف في عناصرها حيث كل عنصر في المصفوقة يمثل سطرا في النص؟

الدالة التالية ستقوم بالمطلوب وتعطي مصفوفة نصية بأسطر الملف:

```
function FileArray(const FileName: string): TStringArray; overload;
var
  Strings: TStrings;
  i: integer;
begin
  result := 0;
  try
    Strings := FileStrings(FileName);
    if Strings <> nil then
    begin
      SetLength(result, Strings.Count);
      for i := 0 to Strings.Count - 1 do
        result[i] := Strings[i];

    end;
  finally
    if Strings <> nil then
      Strings.Free;
  end;
end;
```

الدالة ترجع مصفوفة حيوية dynamic array من نوع TStringArray والذي نحتاج إلى تعريفه قبل استخدامه في بناء الدالة، لذلك يجب وضع التعريف في مكان يكون في منظور الدالة ومنظور من يستدعيها:

```
type
  TStringArray = array of string;
```

وكما في دالة FileText السابقة تقوم الدالة بالاستعانة ب FileStrings لتردّ لها قائمة نصية من نوع TStrings ، وبناء على عدد عناصر القائمة تحدد طول المصفوفة بنفس العدد، ثم تنسخ محتويات كل عنصر في القائمة إلى ما يقابلها من عناصر المصفوفة.

يمكن الآن استخدام هذه الدالة كما في المثال التالي:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  arrItems: TStringArray;
  i: integer;
begin
  arrItems := FileArray('sample.txt');
  Memo1.Clear;
  for i := 0 to High(arrItems) do
    Memo1.Lines.Add(arrItems[i]);
end;
```

وأخيرا

تتعدد المهام البرمجية التي نستخدمها بشكل متكرر في برامجنا، هذه المهام قد يتم تنفيذها في تعليمة واحدة أو في مجموعة متعددة من التعليمات برمجية، تغليف هذه المهام وتعليماتها في تعليمة صغيرة موجزة تزيد من إنتاجيتنا وتحسّن مقروئية برامجنا.

فيما سبق كان عرضا لما يمكن اختصاره أو إيجازه من مهام برمجية، وهناك العديد من المهام الأخرى التي تستحق العمل على إيجازها بالنسبة للمبرمج بحسب عاداته ومجال برمجته.

في ختام مقالتي أودّ الإشارة إلى تعليمة أخرى نسيت ذكرها، تعليمة مزعجة يكثر استخدامي لها؛ هي تعليمة UpperCase . كالعادة، اختصرتها في دالّة U_ :

```
function _U(const s: string): string;
begin
  result := AnsiUpperCase(s);
end;
```

دلفي و ملفات قواعد بيانات MS SQL SERVER - بقلم PLANSHARP

الاتصال قبل/بعد التحزيم والتوزيع

مقدمة:

يحتاج كل مبرمج إلى التعرف على أنواع قواعد البيانات المتاحة و المستخدمة في الأسواق المعلوماتية حتى يتسنى له اختيار المناسب لبرنامجه منها و ذلك يكون حسب مزايا كل منها و مدى تلبيتها للإحتياج.

نتطرق في هذا الموضوع إن شاء الله إلى التعامل بين دلفي و قواعد بيانات MS SQL SERVER, حيث أن هذه الأخيرة هي من أكثر قواعد البيانات إنتشارا وذلك بسبب مرونتها و سهولة التعامل معها و الأهم من ذلك هو توفر ال Driver الخاص بها على العديد من المنصات.

الاتصال بقاعدة البيانات:

من المعروف لدى معظم المبرمجين أن MS SQL SERVER لديها الواجهة الرسومية الخاصة بها لإنشاء قاعدة البيانات و الجدوال الخاصة بها إلى آخره من محتويات, و عند إنشاء قاعدة البيانات فإنه يتولد لدينا ملفان رئيسيان و هما:

- Data File
- Log File

لقواعد بيانات SQL SERVER عدة نسخ و إصدارات حسب عدد المستخدمين و حسب البيئة المراد استخدامها فيها, ومنها أيضاً SQL Server Express وهي نسخة مجانية محدودة ب 10 جيجابايت تخزين لبيانات و عدم وجود SQL Server Agent, و كثيراً ما نستخدمها في برامجنا لما توفره لنا من مزايا.

و كل تلك الإصدارات تعتمد فكرة تواجد قاعدة البيانات على جهاز خادم server بحيث يستقبل الإتصالات من تطبيقات الزبائن و يعالج كل الطلبات.

سنأخذ في عين الإعتبار الإتصال بقاعدة البيانات:

- أثناء التصميم: أي أن الدلفي يعمل و السيرفر موجود في مكان ما على الشبكة المحلية أو على نفس الجهاز, أي أن مكان الملفات معروف تماما.
- بعد التوزيع: أي أن البرنامج الآن أصبح لدى الزبون و قاعدة البيانات يجب أن تكون على سيرفرات الزبون.

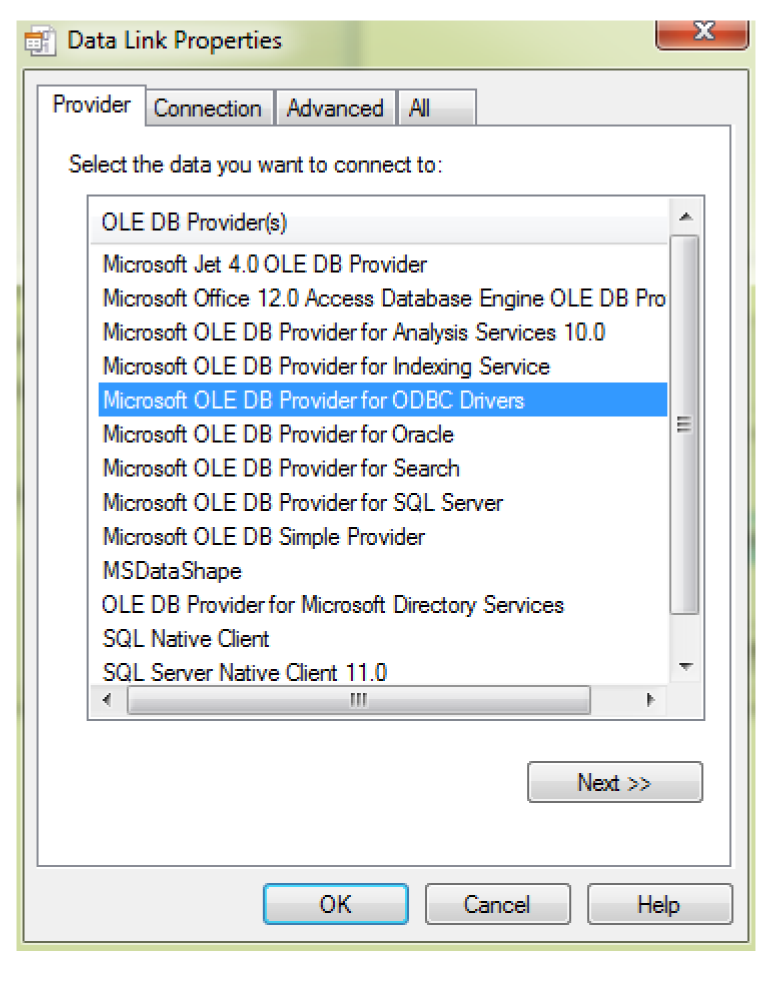
طريقة الإتصال التقليدية وقت التصميم:

كما هو معروف لدى الكثير بأن طريقة الإعتيادية مع قواعد البيانات تتم عن طريق oledb وكمثال على ذلك ADO, لما توفره من سهولة في الإتصال.

عنصر ADOConnection:



باستخدام هذا المكون الموجود ضمن صفحة ADO, نستطيع بناء إتصال مع العديد من قواعد البيانات الشهيرة و التي تستخدم OLEDB Driver, و بالنقر مرتين على هذا العنصر و اختيار build Connection يتوضح لدينا جميع ال drivers المتاحة لنا من أجل الإتصال بقواعد البيانات.



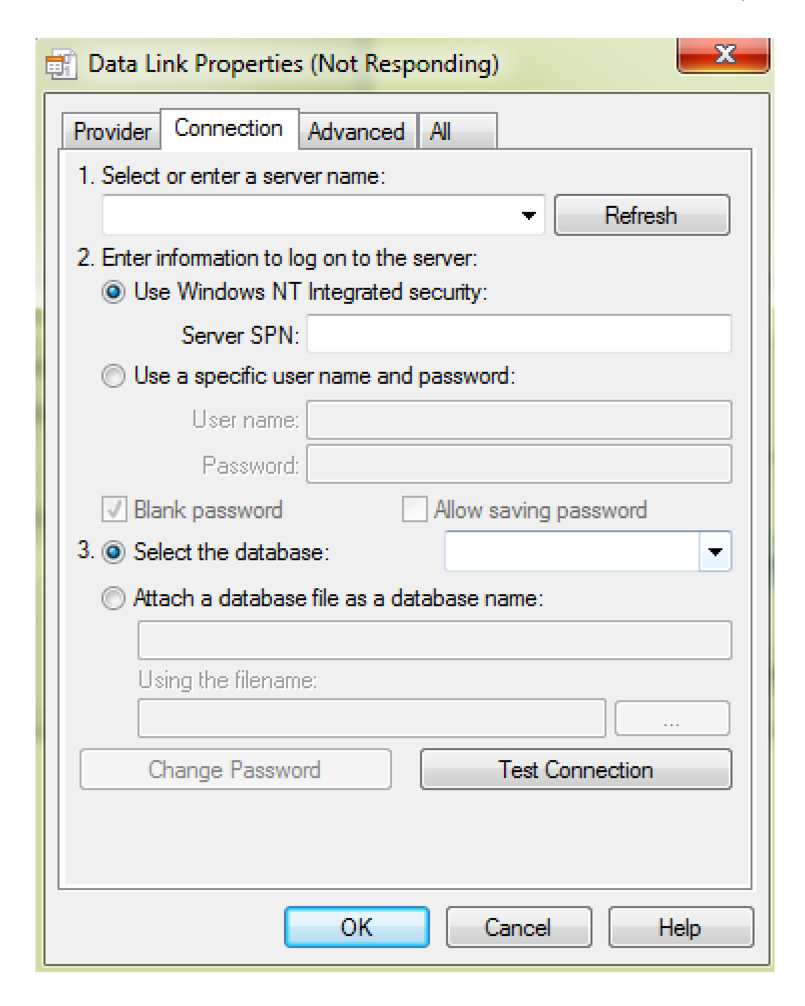
بالنسبة لـ Sql Server:

نحتاج للإتصال بقواعد بيانات SQL Server,

إلى استخدام الـdriver الخاص بالنسخة المستخدمة, و هذا الأخير متوفر على موقع Microsoft, للتحميل من هنا:

https://www.connectionstrings.com/download-sql-server-native-client/

وبعد اختيار الـdriver المناسب, نذهب للنافذة الثانية و فيها يجب تحديد مكان السيرفر و معلومات الإتصال و اسم قاعدة البيانات.



و عند الإنتهاء يتولد لدينا connection string الخاص بالإتصال.

ما هو ال Connection String:هي السلسة المحرفية التي تتضمن جميع متحولات أو بارامترات الإتصال من اسم السيرفر و اسم قاعدة البيانات ونوع الأمان و رقم نسخة الـDriver...إلخ.

مثال عليها:

```
Provider=SQLNCLI11.1;Integrated Security=SSPI;Persist Security
Info=False;User ID="";Password='';Initial Catalog=test;Data
Source=10.10.10.10;Initial File Name="";Server SPN=""
```

طبعاً هنا العملية بسيطة بعد البناء و الإتصال ونحن الآن في التصميم, ولكن ماذا بعد التحزيم و التوزيع؟ أين سيكون السيرفر؟ ماهو اسم المستخدم؟ والباسوورد؟...إلخ من مكونات الإتصال الغير معلومة, مما يجعل قضية بناء مكونات الإتصال من أجل سيرفر الزبون مربك إذا لم نتبع طرق بديلة.

الحل:

يجب من أجل سهولة تعديل الconnection string, هو بناء واحدة ديناميكية وبما أنها string فالقضية بسيطة عندما نعلم ماهي المكونات المتغيرة و الثابتة, أي المكونات التي بحاجة إلى تعديل هي التي ستؤثر في الإتصال.

ملاحظة: لا يوجد طريقة ثابتة لفعل ذلك و إنما هي اجتهادات مبرمجين و كذلك المثال التالي :

بالعودة إلى ال connection string السابق:

```
Provider=SQLNCLI11.1;Integrated Security=SSPI;Persist Security
Info=False;User ID="";Password='';Initial Catalog=test;Data
Source=10.10.10.10;Initial File Name="";Server SPN=""
```

نجد مايلي:

- Data Source: و هي تمثل اسم server/اسم الinstance
- Initial Catalog: و هي اسم قواعد البيانات على السيرفر المختار
- User ID: اسم المستخدم على قواعد البيانات و هي logins
- Password: كلمة مرور المستخدم.

هذه التي ذكرت هي المعرضة للتغيير, لأن ال driver لا يجب أن يتغير لأنه مرتبط بنسخة قاعدة البيانات المستخدمة أثناء التصميم و التي من المفترض أن تون هي نفسها التي عند الزبون.

إذا نستطيع الآن بناء سلسلة ديناميكية من أجل التعديل عليها قبل توزيع البرنامج بسهولة.

الفكرة هي كتابة هذه المتحولات ضمن Listbox مخفية على الفورم الأول من البرنامج و قبل وضع الملف التنفيذي في برنامج التحزيم نقوم بتعديلها.

ولكن قبل ذلك هذه هي المعرفات الثابتة ضمن الكود من أجل بناء ال connection string:

```
const
conn_str1 = 'Provider=SQLNCLI11.1;Persist Security Info=false;User ID=';
conn_str2 = ';Initial Catalog=';
conn_str3 = ';Data Source=';
conn_str4 = ';password=';
```

للتجميع نستخدم مايلي:

```
DataModule1.adoconnectio.ConnectionString := conn_str1
+ListBox1.Items[0] + conn_str2 + ListBox1.Items[1]+ conn_str3 +
ListBox1.Items[2]+conn_str4+ ListBox1.Items[3] ;
```

ال Listbox المخفية على الفورم:

user1
database1
10.10.10.10
userpassword1

وعندما نريد تغيير هذه القيم نستطيع الآن تغييرها بسهولة قبل التحزيم.

استطراد:

يمكن أيضاً وضع تلك المتحولات خارج البرنامج و نقوم بتحميلها أثناء التشغيل و استعمالها, وبهذا نكون قد وفرنا خطوة وهي ReCompile قبل التحزيم, ولكن هذا خطير جداً حيث أن هذه البيانات يجب أن تكون في مكتبة البرنامج و بالتالي فالوصول إليها سهل جدا من قبل المستخدمين فهي معرضة إما للضياع أو للاستخدام غير المشروع.

أيضاً: بالنسبة لوضع متطور قليلاً يمكن عن طريق برامج كثيرة فتح الملف التنفيذي و قراءة هذه المعلومات فهي string وغير مشفرة, فإذا كان من الضروري إخفاؤها نهائياً يجب أن تشفر, و ليس هنا ما يسع ذكر مثال لهذا.

لنكمل: إذا قبل التحزيم أصبح بالإمكان سهولة تغيير هذه المعلومات إلى التي يجب أن تكون عليها عند أجهزة الزبون.

التحزيم:

في حال استخدام قاعدة بيانات ليتم استخدامها من قبل مستخدمين على الشبكةفإن تحزيم التطبيق و استخدامه عند أجهزة الزبائن يحتاج إلى أن تنسخ قاعدة البيانات إلى مكان آمن مثلAppdataأو mydoc, ومن ثم تقوم بعمل attach لها على السيرفر الحالي, كما ،أن التحزيم نفسه من التطبيق يجب أن لا تحتوي على قاعدة البيانات عند توزيعها لدى الزبائن.

إذاً فنحن بحاجة إلى بناء تحزيم للتطبيق و تحزيم لملفات قواعد البيانات:

تحزيم قواعد البيانات:

سنفترض هنا أن هذا التحزيم سيتم تنفيذه مرة واحدة فقط على السيرفر عند بيئة عمل الزبون, ماذا نحتاج لعمل ذلك؟

من السهل وضع ملفات قاعدية البيانات ضمن أحد برامج التحزيم و هي كثيرة ولكن كيف سنربطها مع ال SQL SERVER الموجود على السيرفر, و أين سنضع هذه الملفات.

نحتاج لفعل ذلك إلى عمل attach لتلك الملفات على السيرفر, و هناك العديد من الطرق:

- create database statement
- sys.sp_attach_db
- sys.sp_attach_single_file_db

سنأخذ الطريقة الأولى ونضرب مثال عليها:

```
USE [master]
GO
CREATE DATABASE [test] ON
( FILENAME = N'<path to database file>\test.mdf' ),
( FILENAME = N'<path to log file>\test_log.ldf' )
 FOR ATTACH ;
GO
```

ولكن لتنفيذ هذه التعليمات أثناء التحزيم فنحن بحاجة إلى برنامج يعمل بالخلفية و ينفذ تلك التعليمات و ذلك طبعاً بعد إنتهاء تنفيذ التحزيم, ولتبسيط الأمر سنيتخدم ملف batch لتنفيذ هذه الأوامر وذلك عن طريق استدعاء برنامج SQLCMD, و الذي يفترض أن يكون موجود على السيرفر الذي يحتوي SQL SERVER.

المثال يصبح كالتالي Batch.bat:

```
sqlcmd -S Server\Instance


USE [master]
GO
CREATE DATABASE [test] ON
( FILENAME = N'<path to database file>\test.mdf' ),
( FILENAME = N'<path to log file>\test_log.ldf' )
 FOR ATTACH ;
GO
```

في حال كان السيرفر يحتوي على instance واحد و افتراضي يمكن كتابةipالسيرفر أو اسم السيرفر فقط. عند أول سطر, وطبعا يجب أن يكون لهذا المستخدم الحالي صلاحيات الولوج إلىsql serve.

و هكذا تكون قد نصبت ملفات قاعدة البيانات وتم ربطها مع السيرفر.

تحزيم التطبيق نفسه:

تحزيم التطبيق يحتاج كما هومعلوم إلى تحديد الملفات و المجلدات التي يحتاجها التطبيق أثناء التصميم لتكون معه عند جهاز الزبون, و أهم مافي الأمر و هو محور هذا الموضوع هو ال driver المستخدم للإتصال بقواعد البيانات.

بالنسبة لطريقة تضمين هذا الأخير مع التحزيم, إذا لم تكن هذه الميزة موجودة في برنامج التحزيم الذي تستعمله, فيمكنك أيضا عمل batch file و تنفيذه في أحد مراحل التنصيب.

هذا ال batch file سيحتوي على script تنصيب ال sql server native clientبشكل صامت كالآتي:

```
msiexec /i <filename> /qn IACCEPTSQLNCLILICENSTERMS=YES
```

filename.exeهو يجب أن يكون اسم ملف ال native client الخاص بنسخة قاعدة البيانات المستخدمة.

في حال كان هناك أجهزة 32 bit, وأجهزة 64 bit, لدى العميل فيجب إختيار الdriver المناسب, ولكن من أجل تحزيم واحد قم بعمل سطرين في الملف الدفعي أو ال batch file كالتالي:

```
msiexec /i <filename(32bit version)> /qn IACCEPTSQLNCLILICENSTERMS=YES
msiexec /i <filename(64bit version)> /qn IACCEPTSQLNCLILICENSTERMS=YES
```

في هذه الحالة فإن نظام التشغيل لن يقوم بتنصيب ال driver الغير مناسب, و سيأخذ ال driver المناسب ☺.

كل ماسبق ذكره هو لقواعد بيانات تعمل على الشبكة, وبقي هناك نسخة تدعى LOCALDB, وهو عبارة عن برنامج سيرفر يقوم بعمل hosting لملفات ال Sql Server, لكي تعمل على جهاز واحد فقط و ليس للإستخدام الشبكيو وهو مجاني و لكن محدود أيضاً بـ4 جيجابايت حجم تخزين لملفات قاعدة البيانات, كما أنه لا يحتوي على visual studio.

لتحزيم تطبيق يستخدم هذا النوع LOCALDB, فنحن بحاجة إلى تحزيم ملفات التطبيق و نسخة السيرفر الخاص بنظام التشغيل 32bit/64bit, و أخيرا نحتاج إلى ال driver الخاص بنظام التشغيل لدى الزبون 32bit/64bit.

بعد وضع كل الملفات السابقة ضمن برنامج التحزيم المختار, سنقوم باستخدام نفس الطريقة السابقة بخصوص نوع نظام التشغيل حيث هو سيختار النسخة المناسبة له من السيرفر و ال driver.

ولكن قبل ذلك يجل أن لا ننسى تعديل ال connection string وذلك لأنه لن نقوم باستخدام اسم أو عنوان سيرفر بل سيتم استخدام LOCALDB.

مثال على connection string يستخدم LOCAL DB.

```
Provider=SQLNCLI11.1;Integrated Security=SSPI;Persist Security
Info=False;User ID="";Initial Catalog="";Data
Source=(localdb)\v11.0;Initial File Name=' +  '<path of
mdf>\<name>.mdf;Server SPN="";
```

طريق تحزيم السيرفر و ال driver في هذه الحالة تكون:

```
msiexec /i <filename(32 bit version)> /qn IACCEPTSQLLOCALDBLICENSETERMS=YES
msiexec /i <filename(64 bit version)> /qn IACCEPTSQLLOCALDBLICENSETERMS=YES
msiexec /i <filename(32 bit version) > /qn IACCEPTSQLNCLILICENSETERMS=YES
msiexec /i <filename(64 bit version)> /qn IACCEPTSQLNCLILICENSETERMS=YES
```

رابط تحميل ال LOCAL DB:

http://microsoft-sql-server-express-localdb.software.informer.com/download/

ودمتم بخير...

عدة إصدارات من مشروع واحد في Delphi / Lazarus - بقلم Lam.Abdeldjalil

تريد إنشاء عدة إصدارات من برنامجك (إصدار خفيف / كامل / تجريبي....) ، و ليس لديك خبرة كبيرة في الحماية . وتريد عمل ذلك بدون الاستعانة بأدوات خارجية و في مشروع واحد فقط ! إذا كان هذا مرادك استعد لتعلم "conditional compilation"

ملاحظة: هذا الموضوع ليس خاص بحماية البرامج .

conditional compilation معروفة عند كل من مطوري المكونات و مطوري البرامج المتعددة المنصات .

قبل البدء يجب أن نعرف ما هي conditional compilation (الترجمة الشرطية)

تمنحنا conditional compilation اختيار أجزاء من الكود و ترجمتها "compile " بشرط وضعناه ، الجزء المحقق لهذا للشرط يترجم ويضمن في الملف التنفيذي أما الجزء الغير المحقق لهذا للشرط لا يترجم و لا يضمن.

$IfDef تبدأ شرط الـ conditional compilation

$Else في حال لم يتحقق الشرط . تكون في هذا السياق :

```
{$IfDef DefName}
 //  الجزء الأول
 {$Else}
// الجزء الثاني
 {$EndIf}
```

DefName تسمى الرمز الشرطي 'conditional symbol ' و توجد قائمة من الرموز الشرطية يمكن القول محجوزة منها

```
{$IFDEF MSWINDOWS}//نظام ويندوز
{$ENDIF}
{$IFDEF VER140}// إصدار المترجم 14 أي دلفي 6
{$ENDIF}
```

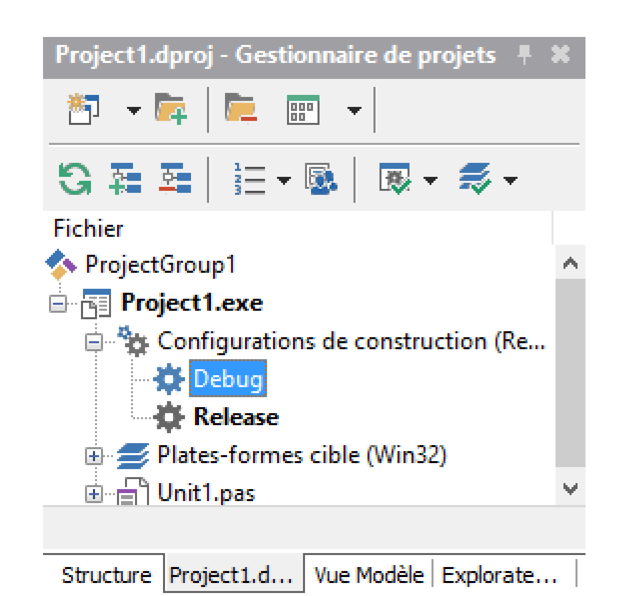
تجد القائمة في الرابط

انتبه الرموز الشرطية لا تستعمل مع if...then/else .

* للتوضيح أكثر أريدكم أن تنشئوا مشروع جديد (مرئي) ضع Label ، في الحدث OnCreate ضع الكود

```
{$IFDEF DEBUG}
  Label1.Caption := 'this is debug mode';
{$ELSE}
  Label1.Caption := 'this is release mode';
{$ENDIF}
```

الآن قم باختيار البناء Debug (اضغط عليها مرتان) ثم ترجم F9 ، لاحظ أن نص الـ Label قد تغير حسب الشرط الموضوع . جرب نفس الشيء مع Release و لاحظ الفرق .



لآن نتطرق إلى تفاصيل أكثر من هذا المثال البسيط :

سوف ننشئ برنامج صغير يحتوي على 3 إصدارات خفيف LIGHT، كامل FULL و تجريبي DEMO .

الكامل يحتوي على 3 نوافذ ، الخفيف يحتوي على نافذتين و الثالثة ليست مضمنة , أما التجريبي نافذة واحدة و الباقي ليست مضمنة.

- أنشئ مشروع جديد فيه 3 نوافذ
- ضع زرين Buttons في النافذة الرئيسية ثم أنسخ هذا الجزء في الحدث OnCreate

```
{$IFDEF DEMO}
  Caption:='إصدار تجريبي';
  Button1.Enabled:=false;
  Button2.Enabled:=false;
{$ELSE}
{$  IFDEF LIGHT}
    Caption:='إصدار خفيف';
    Button1.Enabled:=true;
    Button2.Enabled:=false;
{$  ELSE}
    Caption:='إصدار كامل';
    Button1.Enabled:=true;
    Button2.Enabled:=true;
{$  ENDIF}
{$ENDIF}
```

- الزران مسئولان عن إظهار النوافذ الأخرى ، قمنا في الحدث السابق تفعيل / إلغاء تفعيل الأزرار حسب الإصدار.
- نقم الآن بتحديد الصلاحيات الخاصة بكل إصدار
- نحدد لكل مشروع الوحدات المضمنة في uses الخاصة بالوحدة الرئيسية

```
$}IFDEF FULL{
 // كل الوحدات في الكامل uses Unit2, Unit3;
$}ELSE{
$}  IFDEF LIGHT{
    // الوحدة الثانية فقط في الإصدار الخفيف uses Unit2;
$}  ENDIF{
$}// التجريبي لم نضمن أي شيئ ENDIF{
```

- نحدد لكل مشروع الوحدات المضمنة في uses الخاصة بالمشروع في Project Source

```
uses
  Vcl.Forms,
  Unit1 in 'Unit1.pas' {Form1{
$} IFDEF FULL{
,  Unit2 in 'Unit2.pas' {Form2,{
  Unit3 in 'Unit3.pas' {Form3{
$} ELSE{
$}  IFDEF LIGHT{
,   Unit2 in 'Unit2.pas' {Form2{
$}  ENDIF{
$} ENDIF{
```

- في حالة الإنشاء الأوتوماتيكي للنوافذ يجب تحيدها كذلك

```
  Application.CreateForm(TForm1, Form1;(
$} IFDEF FULL{
  Application.CreateForm(TForm2, Form2;(
  Application.CreateForm(TForm3, Form3;(
$} ELSE{
$}  IFDEF LIGHT{
   Application.CreateForm(TForm2, Form2;(
$}  ENDIF{
$} ENDIF{
```

- الآن نضع الترجمة الشرطية الخاصة بالأزرار في الحدث OnClick الأول

```
$}IFDEF FULL{
  form2.Show;
$}ELSE{
$}  IFDEF LIGHT{
    form2.Show;
$}  ENDIF{
$}ENDIF{
```

الثاني

```
$}IFDEF FULL{
  form3.Show;
$}ENDIF{
```

Project1.dproj - Gestionnaire de projets
Fichier
Configurations de construction (Re...
Debug
Release
Make
Construire
Nettoyer
Nouvelle configuration...
Nouveau jeu d'options...
Appliquer le jeu d'options...
Renommer
Supprimer
Activer
Modifier...

1 ـ الآن نقوم بالتهيئة للبناء / البناء :

أ ـ الدلفي:

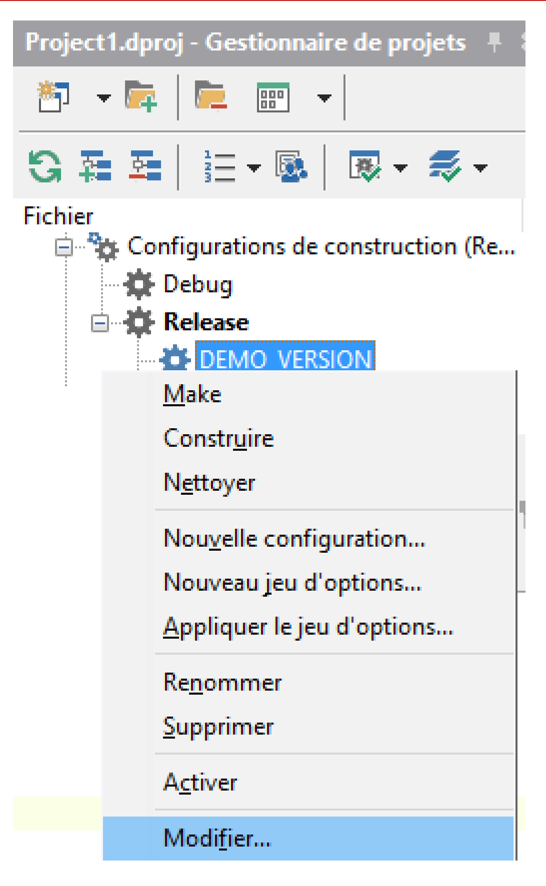
اضغط بالأيمن على RELEASE ثم

Nouvelle / New

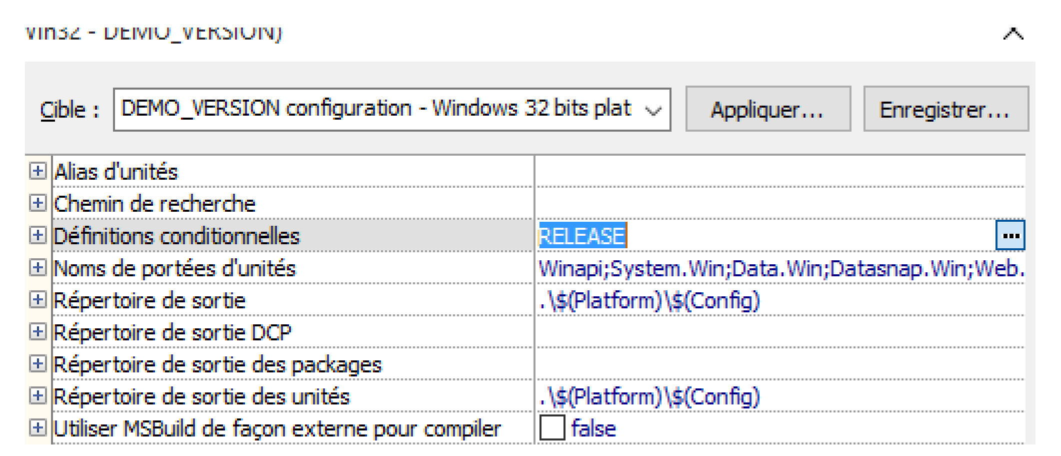
ثم أدخل الثلاث إصدارات الخاصة بنا .

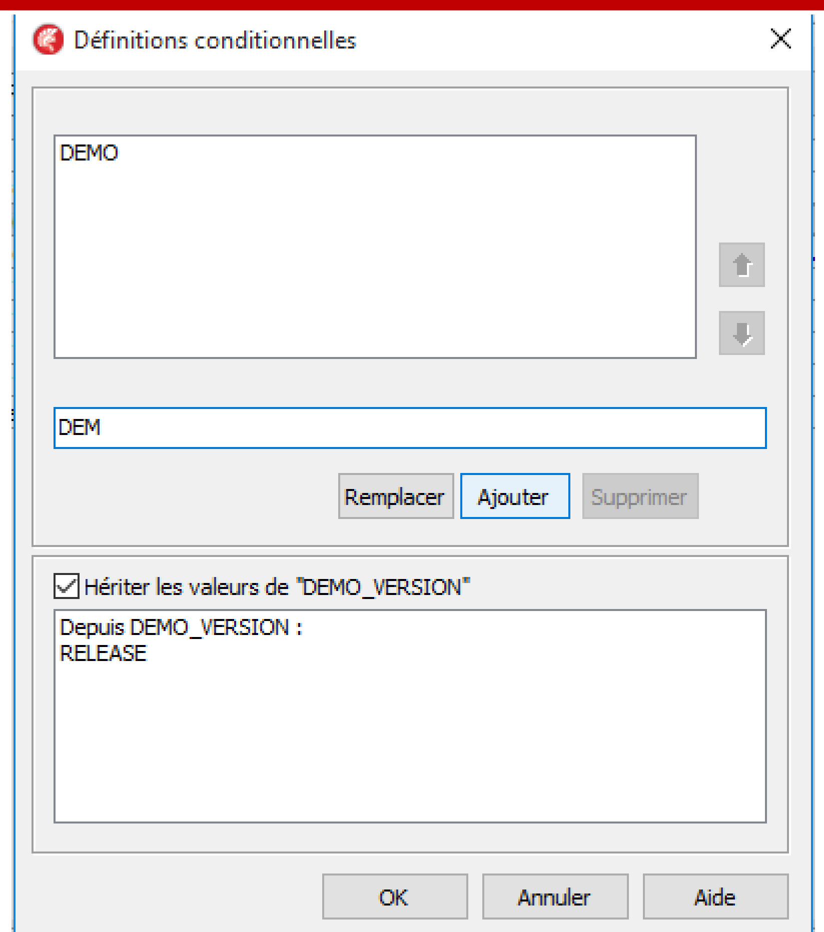
Project1.dproj - Gestionnaire de projets
Fichier
Configurations de construction (Re...
Debug
Release
DEMO_VERSION
FULL_VERSION

Configuration de construction
Entrez le nom de la nouvelle configuration de construction : LIGHT_VERSION
OK
Annuler



- بعدها اضغط باليمين على التهيئة ثم Modifier / Edit
- ستفتح لك نافذة Options de Projet /Project options
- في حالة DEMO_VERSION أضف DEMO إلى قائمة Définitions Conditionnels / Conditional defines
- ثم اضغط على ok

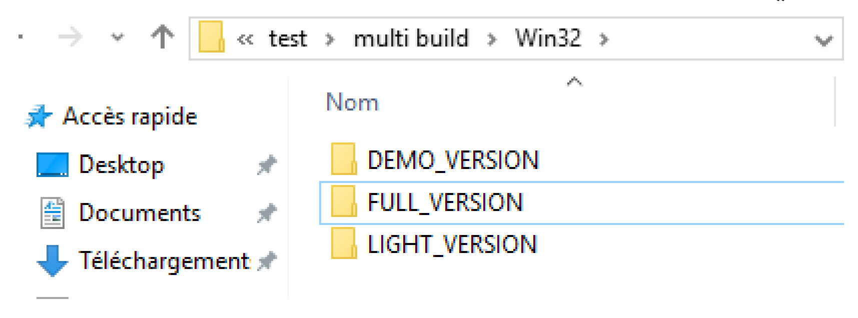




- أعد نفس الخطوات مع FULL_VERSION أضف FULL و مع LIGHT_VERSION أضف LIGHT

2 ـ اضغط مرتين على الإصدار المراد بنائه ثم ترجم F9

تكون المخرجات كما في الصورة

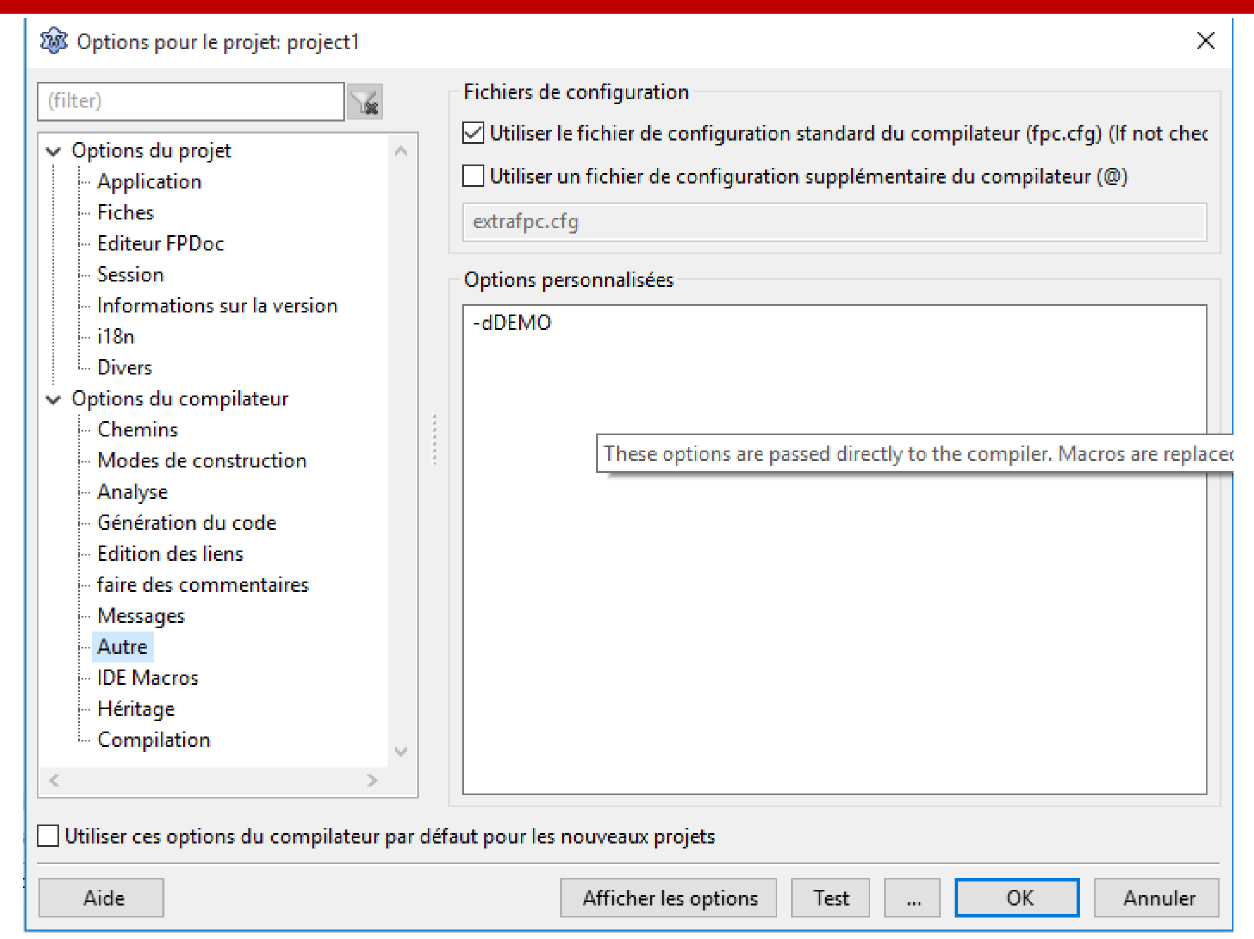


ب - Lazarus :

في Lazarus نحدد مباشرة الإصدار المراد بنائه بـ : -dver

ver الإصدار مثلا : -dDEMO

Options->Project Options->Compiler Options->Additional compiler options



ملاحظة:

انتبه للـ uses و Application.CreateForm عند استعمالها بين شروط الـ conditional compilation في بعض الأحيان يقوم الـ IDEبإفسادها عند تغيير الإصدار .

المثال مرفق (الخاص بالدلفي افتح بالملف ProjectGroup1.groupproj). بالتوفيق.

تعرف على دوال النظام GetWindowLong/ SetWindowLong - بقلم Agmcz

سنقوم في هذا الموضوع بعمل شرح للدالتين GetWindowLong و SetWindowLong

مع ثلاث حالات

GWL_EXSTYLE

GWL_STYLE

GWL_WNDPROC

الدالتين يتعملان مع النوافذ Window بشكل عام

قبل البدأ في استعمال الدالتين لابد من وجود Control للتطبيق عليه. لذا سننشأ واحد يلبي حجتنا. بما أننا نتكلم عن دوال API سنقوم بإنشاء Edit مباشرة بإستعمال دوال API بالتحديد الدالة CreateWindowEx

لتفاصيل أكثر يمكنك تصفح رابط الدالة من موقع ms

https://msdn.microsoft.com/en-us/library/windows/desktop/ms632680%28v=vs.85%29.aspx

للدالة 12 برامتر

1 dwExStyle أو Extended Window Styles

وهو نمط يمس كل نوافذ بشكل عام وهذا رابط كل الثوابت التي يمكن استخدامها من موقع ms

https://msdn.microsoft.com/en-us/library/windows/desktop/ff700543%28v=vs.85%29.aspx

2 lpClassName

3 lpWindowName

4 dwStyle أو Window Styles

https://msdn.microsoft.com/en-us/library/windows/desktop/ms632600%28v=vs.85%29.aspx

5 6 7 8 خاصة بإحداثيات Window

9 وهو هندل النافذة الأب

10 في حال وجود Menu يمكن تمرير مقبضها لربطها مع نافذة

11 ....

12 برامتر اضافي من نوع مؤشر .

بما أننا نتكلم عن EDIT يوجد أيضا ما يسمى Edit Control Styles وهي أنماط خاصة بهذا Control. لمعلومات أكثر

https://msdn.microsoft.com/en-us/library/windows/desktop/bb775464%28v=vs.85%29.aspx

يتم دمجها مع Window Styles

نعود للدالة: الدالة تقوم بإرجاع قيمة وهي مقبض نافذة Edit في حالة كانت النتيجة صفر يعني أنه لم يتم إنشاء Edit

هذا مثال لانشاء مكون

```
var
  hEdit: THandle;
begin
  hEdit := CreateWindowEx(
    0,
    'EDIT',
    nil,
    ES_LEFT or
    ES_AUTOHSCROLL or
    WS_CHILD or
    WS_VISIBLE,
    32, 8, 337, 21, Handle, 0, HInstance, nil);
end;
```

بعد التنفيذ النتيجة

الـ Edit بنمط مختلف عن الإعتيادي بسبب اسناد القيمة صفر للبرامتر الأول

لذا سنقوم بتعديل Edit في وقت التنفيذ وإسناد ExStyle بالتحديد WS_EX_CLIENTEDGE لنحصل على الشكل المناسب

لذا سنلجأ للدالة SetWindowLong

للدالة ثلاث مدخلات

1 الهندل

2 نوع الخدمة

3 القيمة الجديدة

لإسناد قيمة جديدة يجب الحصول على القيمة القديمة واضافة لها الجديدة

التجميع تم باستمعال المعامل OR

لذا سنستعمل الدالة GetWindowLong

للدالة برامترين

الأول هندل Edit

الثاني نوع الخدمة

الدالة ترجع Old Long أو النمط الحالي المستخدم

أيضا للعلم الدالة SetWindowLong عند الإسناد ترجع أيضا Old Long

أو Prev Long

التمرير سيكون بهذا الشكر

```
var
  dwOldLong: DWORD;
begin
  dwOldLong := GetWindowLong(hEdit, GWL_EXSTYLE ;(
  SetWindowLong(hEdit, GWL_EXSTYLE, dwOldLong  or WS_EX_CLIENTEDGE;(
end;
```

بعد تنفيذ الكود لا تغير لحل المشكل نستعمل الدالة SetWindowPos

بهذا الشكل

```
SetWindowPos(hEdit, 0, 0, 0, 0, 0, SWP_FRAMECHANGED or SWP_NOMOVE or
SWP_NOSIZE;(
```

لعمل update لنحصل على

بما أننا وضحنا كيفت اسناد نمط جديد حتى بعد انشاء Control

واستخدمنا الدالتين GetWindowLong SetWindowLong

سنحاول تفكيك في حالات Style

بما أن التجميع تم بواسطة التعليمة OR

سنعيد الفك للحصول على STYLE المستخدمة

لقد قمت بكتابة وحدة تقوم بذلك ووضعتها في المنتدى لذا سنستعملها

الوحدة مرفقة

بها ربع دوال

```
procedure GetWindowStylesList(hWindow: THandle; out StylesList:
TStringList;(
procedure GetWindowStylesExtendedList(hWindow: THandle; out
StylesList: TStringList;(
function WS_Exists(hWindow: THandle; Style: Integer): Boolean;
function WS_EX_Exists(hWindow: THandle; ExStyle: Integer): Boolean;
```

الدالتين

GetWindowStylesList

GetWindowStylesExtendedList

يرجعان قائمة بـ Styles المستخدمة

أما الدالتين WS_Exists و WS_EX_Exists فهما للتحقق من وجود Style

```
var
  SL, SL2: TStringList;
begin
  SL := TStringList.Create;
  SL2 := TStringList.Create;
  try
    GetWindowStylesList(hEdit, SL;(
    GetWindowStylesExtendedList(hEdit, SL2;(
    Memo1.Text := SL.Text;
    Memo2.Text := SL2.Text;
  finally
    SL.Free;
    SL.Free;
  end;
end;
```

بعد التنفيذ النتيجة

WS_OVERLAPPED
WS_CHILD
WS_VISIBLE

WS_EX_CLIENTEDGE
WS_EX_LEFT

الآن سنوضح الحالة الأخيرة GWL_WNDPROC و من خلالها سنخصص إجراء خاص للـ Contolلمعالجة الأحداث أي تخصيص WindowProc

سنقوم باصتياد الحدث الذي تكون فيه الماوس تتحرك فوق Control

بالتحديد الحدث WM_MOUSEMOVE واضهار رسالة مرحبا

أولا نعرف متغير من نوع مؤشر Global

```
var
  PrevWndProc: Pointer;
```

سأذكر دوره لاحقا مع إضافة هذا الإجراء

```
function EditWindowProc(hWnd, uMsg: UINT; wParam: WPARAM; lParam:
LPARAM:(
  Integer; stdcall;
begin
  Result := 0;
  case uMsg of
     WM_MOUSEMOVE:
      begin
        ShowMessage('Hello;('!
      end;
  else
    Result := CallWindowProc(PrevWndProc, hWnd, uMsg, wParam, lParam;(
  end;
end;
```

الأن سنقوم بربط الحدث مع Edit باستعمال SetWindowLong

```
PrevWndProc := Pointer(SetWindowLong(hEdit, GWL_WNDPROC,
Integer(@EditWindowProc;)((
```

ننفذ الكود ونذهب فوق Edit نلاحظ الإستجابة



بما أننا ذكرنا سابقا أن ماترجعه الدالة هو Old Long

الدالة ترجع لنا عنوان Old WindowProc

لاحظ أننا استعملنا الدالة CallWindowProc

لإعادة التوجيه في حالة كان Msg مجهول واعادة التوجيه إلى Old WindowProc

المتباينات أو المتقلبات (Variants) في دلفي - بقلم لؤي

المتباينات بحسب الترجمة أو الفهم هي نوع من المتغيرات التي يدعمها دلفي:

لتقديم الدعم الكامل OLE، يتضمن إصدار 32 من دلفي نوع متغير من البيانات. هنا نود التحدث عن هذا النوع من البيانات بشكل عام. نوع متباين، في الواقع، أصبح له تأثير متزايد على كامل اللغة، ويستخدم مكونات دلفي أيضا بطرق ليست لها علاقة ببرمجة OLE.

المتباينات ليس لها نوع :

بشكل عام، يمكنك استخدام المتباينات لتخزين أي نوع البيانات وإجراء العمليات و عدة تحويلات. لاحظ أن هذا يتعارض مع النهج العام للغة باسكال و ضد البرمجة الجيدة.

والمتباين هو نوع يختبر وينفذ من طرف النظام في وقت واحد. أي أن الكومبيلر (المترجم) لن يحذرك من الأخطاء المحتملة في كود المبني ، والتي يمكن أن تتكتشفها إلا باختبارات مكثفة.

على العموم، يمكنك أن تعتبر الأكواد التي تستعمل المتغيرات بأنها مترجمة مسبقا لأن لا يمكن أن تنفيذ أي عملية حتى وقت التشغيل. وهذا يؤثر بشكل خاص على سرعتها البرمجية.

و الآن، لقد حذرتكم ضد استخدام هذا النوع من المتباينات، حان الوقت للنظر في ما يمكن القيام به

وبمجرد تعريف متغير من نوع متباين كما يلي:

```
var
  V: Variant;
```

كما يمكنك أن تخصص له عدة أنواع مختلفة:

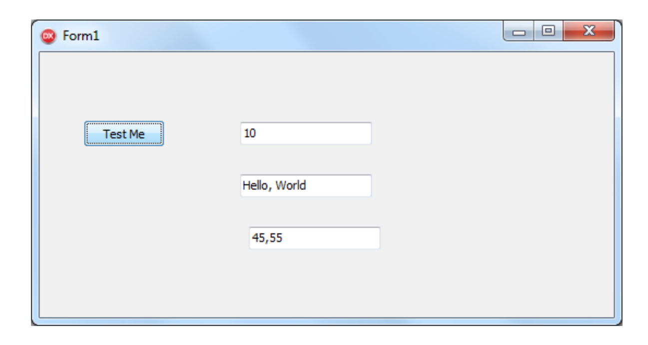
```
V := 10;
V := 'Hello, World';
V := 45.55;
```

وبمجرد إعطاء قيمة للمتباين، يمكنك نسخ قيمته إلى أي نوع متوافق أو غير متوافق إذا قمت بتعيين القيمة إلى نوع بيانات غير متوافق، فإن دلفي يقوم بإجراء التحويل، إذا استطاع ذلك. وإلا سيصدر خطأ وقت التشغيل ومنه نستنتج أنه يمكن تشغيل أي عملية بدون النظر إلى كل محتوى الكود و يبقى ذلك خطر وغير آمن

بالنظر في المثال التالي (اسمه VariTest)، الذي يعد امتداد للتعليمات البرمجية أعلاه. وضعنا ثلاثة Edit على شكل جديد، و اضفنا بعض الأزرار، ثم الكود التالي للحدث OnClick للزر الأول:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  V: Variant;
begin
  V := 10;
  Edit1.Text := V;
  V := 'Hello, World';
  Edit2.Text := V;
  V := 45.55;
  Edit3.Text := V;
end;
```

مضحك أليس كذلك ؟ بالإضافة إلى ذلك يمكن من الكود أعلاه أن لك أن تضع ما احببت بنفس المتغير V من نوع متباين عدد صحيح ثم متغير نصي و كذلك عدد حقيقي الناتج في الصورة أدناه (الصورة رقم 1)



الصورة رقم 1

والأسوأ من ذلك، يمكنك استخدام المتباينات لحساب القيم، وكما هم موضح في الزر الثاني:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  V: Variant;
  N: Integer;
begin
  V := Edit1.Text;
  N := Integer(V) * 2;
  V := N;
  Edit1.Text := V;
end;
```

كتابة هذا النوع من التعليمات البرمجية محفوف بالمخاطر، على أقل تقدير. إذا احتوى Edit على عدد، كل شيء يعمل. ، و إلا سيحدث خطأ.

ومايلي لائحة كاملة من أنواع المتباينات المتاحة:

```
varArray
varBoolean
varByRef
varCurrency
varDate
varDispatch
varDouble
varEmpty
varError
varInteger
varNull
varOleStr
varSingle
varSmallint
varString
varTypeMask
varUnknown
varVariant
```

يمكنك أن تجد شرح لهذه الأنواع في المتغيرات في دلفي مساعد 1F.

المتباينات بطيئة :

الكود الذي يستخدم نوع متباين سيكون بطئ، ليس فقط عند تحويل نواع البيان، ولكن أيضا عند إضافة قيمتين لمتباينين يحملان عددين صحيحين .سيكون التنفيذ بطيئ مثل كود الـ Visual Basic لمقارنة سرعة خوارزمية مبنية على المتباينات مع أخرى مبنية على متغيرات من الأعداد الصحيحة، يمكنك إلقاء نظرة على مثال السرعة.

على زر بإستخدام المتباينات:

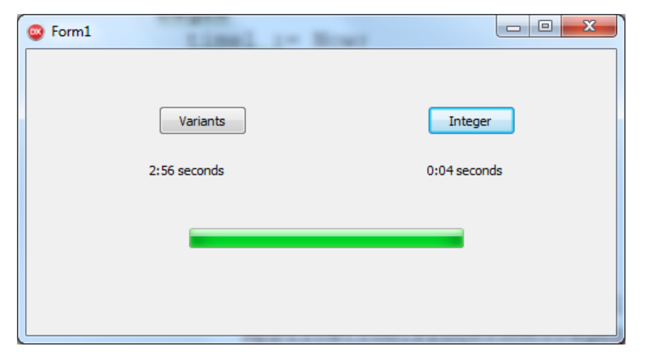
```
procedure TForm1.Button1Click(Sender: TObject);
var
  time1, time2: TDateTime;
  n1, n2, Total: Variant;
begin
  time1 := Now;
  n1 := 0;
  n2 := 0;
  ProgressBar1.Position := 0;
  while n1 < 500000000 do
  begin
    n2 := n2 + n1;
    Inc (n1);
    if (n1 mod 50000) = 0 then
    begin
      ProgressBar1.Position := n1 div 50000;
      Application.ProcessMessages;
    end;
  end;
  // we must use the result
  Total := n2;
  time2 := Now;
  Label1.Caption := FormatDateTime ('n:ss', Time2-Time1) + ' seconds';
end;
```

على زر 2 بإستخدام متغير من نوع صحيح:

```
procedure TForm1.Button2Click(Sender: TObject);
var
  time1, time2: TDateTime;
  n1, n2, Total: Integer;
begin
  time1 := Now;
  n1 := 0;
  n2 := 0;
  ProgressBar1.Position := 0;
  while n1 < 500000000 do
  begin
    n2 := n2 + n1;
    Inc (n1);
    if (n1 mod 50000) = 0 then
    begin
      ProgressBar1.Position := n1 div 50000;
      Application.ProcessMessages;
    end;
  end;
  // we must use the result
  Total := n2;
  time2 := Now;
  Label1.Caption := FormatDateTime ('n:ss', Time2-Time1) + ' seconds';
end;
```

سيمكنك رؤية الفرق بين السرعتين أثناء التنفيذ و يختلف ذلك من حاسوب إلى آخر فمثلا على حاسوبي نفس الكود أخد أكثر من دقيقتين للإنتهاء من التنفيذ وأقل من 5 ثواني بإستعمال المتغيرات !!!! أنظر الصورة أدناه:





سرعات مختلفة من نفس الخوارزمية، بناء على الأعداد الصحيحة والمتباينات (التوقيت الفعلي يختلف بإختلاف الكمبيوتر)، كما يتضح من المثال VSpeed.

خلاصة:

المتباينات مختلفة عما هي عليه مقارنة بلغة الباسكال الكلاسكية التي حاولنا التطرق إليها بصورة موجزة. وعلى الرغم من دورها في برمجة OLE، فإنها يمكن أن تكون في متناول اليد لكتابة برامج سريعة وسيئة دون الحاجة حتى إلى التفكير في أنواع البيانات. كما رأينا، و هذا يؤثر على الأداء إلى حد بعيد.

مترجم بتصرف عن:

Marco Cantù's Essential Pascal Chapter 10 : Variants

استعمال الأحداث Events - بقلم unprogramme

تحديث البيانات تلقائيا في قاعدة Interbase/FireBird

أنا أنهيت برنامجي للتسيير باستعمال قاعدة بيانات Interbase/FireBird، برنامجي يعمل جيدا على الشبكة، لكنني أواجه مشكلة تحديث عرض البيانات في جميع الأجهزة المتصلة في نفس الوقت، كيف السبيل إلى ذلك؟

كم هو كبير عدد الأسئلة التي تصب في نفس المضمار ، والحل بسيط إن شاء الله، وهو يكمن فاستعمال بما يعرف بالأحداث أي Events.

دعونا ندخل في التطبيق مباشرة...بالنسبة لقاعدة البيانات Interbase/Firebird

لنفرض أن في قاعدة البيانات هناك جدول يحوي قائمة البضائع Articlesاو Goods ، فلابد أنه في لحظة ما هناك من يقوم بإضافة بضاعة جديدة ، أو هناك من يقوم بتغيير الكمية عن طريق عملية البيع و الشراء ، و أيضا هناك من يقوم بعرض القائمة فقط .

فكيف لهؤلاء الأشخاص أن علموا بالتغييرات التي حدثت على هذا الجدول في النفس الوقت.

أولا نقوم بتسجيل الأحداث التي تطرأ على هذا الجدول ، وذلك باستعمال Triggers

لدينا ثلاث حالات :

1- في حالة إضافة بضاعة جديدة أو منتوج جديد :

نكتب الحدث التالي :

```
SET TERM; ^
CREATE TRIGGER  POST_New_Goods FOR  MyGoods
ACTIVE AFTER INSERT POSITION 0
AS
BEGIN
    POST_EVENT 'New_Goods' ;
END
^
SET TERM^;
```

2- في حالة تغيير قيمة السجل من جدول البضائع .

نكتب الحدث التالي :

```
SET TERM; ^
CREATE TRIGGER POST_MOD_GOODS FOR MyGoods
ACTIVE AFTER UPDATE POSITION 0
AS
BEGIN
    POST_EVENT 'Modif_Goods' ;
END
^
SET TERM^;
```

3- في حالة حذف سجل من جدول البضائع :

نكتب الحدث التالي :

```
SET TERM; ^
CREATE TRIGGER POST_DEL_GOODS FOR MyGoods
ACTIVE AFTER DELETE POSITION 0
AS
BEGIN
    POST_EVENT 'Delete_Goods' ;
END
^
SET TERM^;
```

بالنسبة للبرمجة في الدلفي :

قم بجلب المكون IBEvents تجده في التبويب Interbase بالنسبة لمن يستعمل مكونات ال IBX

قم بالعمليات التالية .

أولا : لابد أن تحدد الأحداث التي تنتظرها ، فمثلا في نافذة قائمة البضائع أن لا تنتضر التحديث الذي يطرأ على جدول الموردين أو الزبائن ، و إنما تنتظر فقط الأحداث التي تطرأ على جدول البضائع.

في الحدث Create للنافذة قم بكتابة قائمة الأحداث التي تنتظرها :

```
IBEvents1.UnRegisterEvents;
IBEvents1.Events.Add('New_Goods') ;
IBEvents1.Events.Add('Modif_Goods') ;
IBEvents1.Events.Add(' Delete_Goods');
IBEvents1.Registered := true;
```

و هكذا قمنا بتسجيل جميع الأحداث المنتظرة.

جميل ....

يبقى أن نعرف أي الأحداث التي طرأت ؟

في الحدث OnEventAlert نقوم بالتقاط الأحداث و إجراء العملية المناسبة لتحديث عرض البيانات.

```
If EventName = 'New_Goods' then
  Var Bmk : TBookMark;
begin
Bmk := MyGoodstable.GetBookmark;

MyGoodstable.Close;
MyGoodstable.Open;
MyGoodstable.GotoBookMark(Bmk);
FreeBookmark(Bmk);

end;
```

أو عوضا عن ذلك للتجربة نكتب عوض عن الكود السابق الكود التالي :

```
if MyGoodstable.State = dsBrowse then
begin
ShowMessage('a New Goods Insert');
end
```

و هكذا نفس الإجراء لجميع الأحداث ...

لمحات لطريق البرمجة - بقلم فريد

أول ما يجب معرفته هو أن تضع صوب عينيك أن الأمر ليس بالسهولة و لا بالصعوبة. الأمر كله متوقف عليك و على مدى تمسكك بهدفك. و اعلم أن الله عز و جل لن يغير من شأنك شيء ما لم تغيره أنت بذاتك.

-سل نفسك : ماذا تريد من البرمجة؟ ما هو هدفك, طموحك؟ ماذا أعددت لتحقيق ما تريد ؟إلى ما تسعى ؟ ما هي خارطة طريقك ؟ إذا لم تجد إجابة أو أنك لم تقتنع فالأولى لك استغلال هذا الوقت الثمين فيما يرجع عليك بالنفع و الفائدة لا أن تضيّعه في محاولة إدراك ما لست أنت بداركه.

- اعلم -رحمك الله- أن الطريق إلى الاحتراف في البرمجة له ظله, يشبهه و يماثله إلى درجة تصعب عليك التعرف على من هو ذا و من هو ذاك. فإما أن تتخصص في البرمجة بلغة معينة و تحاول التعمق فيها و الاحتراف بها و الإلمام بجوانبها و خفاياها و صقل مهارتك بها أو أن تحاول الأخذ من هذا و التذوق من ذاك و شم الآخر ثم ترك الأول و الجري وراء الثاني و في النهاية لا عم و لا خال!.

دوّن ملاحظات, قصاصات كود, معلومات برمجية, أفكار, لمحات ... الخ اجعل لنفسك كراسا أو ملف وورد خاصا لذلك ليسهل عليك الحفاظ عليها و الرجوع لها حين تحتاجها. ألم تر إلى قول القائل : العلم صيد و الكتابة قيده.

-اقرأ الكتب و الكتب و الكثير من الكتب, أو الدورات و الفيديوهات التعليمية, و اعلم أن في وقتنا الحاضر العلم أوفر من أي وقت مضى و هو أقرب من طالبه حتى و كأن حاله يقول : إلا من أبى.

- التجربة و الخبرة : التجربة هي الجهد و الخبرة في المكافئة. هناك مقولة عند المبرمجين تقول أنه إذا عمل البرنامج من أول محاولة فأنت على خطأ, لذلك عليك أن تجرب المرات العديدة بل الكثيرة لتصل إلى مبتغاك. و هنا اختبار للصبر و امتحان للمثابرة.

- النظام : الإنسان المنتظم في حياته تعرفه من أول وهلة أو طلة على حاسوبه أو مشروعه, فتراه قد نظم ملفاته بعناية داخل مجلدات خاصة معنونة بدقة بل أنظر إلى مشروعه أو برنامجه, حيث قام بتنظيم وحدات المشروع جيدا و بشكل يسهل عليه تحليل الأخطاء فيما بعد أو حين العمل على تحديث البرنامج أو تحسينه و قد رتب الأكواد المصدرية بطريقة منسقة و أضاف تعليقات هنا و هناك ليسهل عليه تذكر النقاط المهمة في الكود, ثم ربما فصل بعض قطع الكود المتشابهة في الدور الذي تؤديه في وحدات مستقلة ليتجنب كتابة نفس الكود مرات عديدة, هذه وحدة الحسابات و تلك وحدة قواعد البيانات و الأخرى للشبكة و هكذا.

- التخطيط للبرنامج و وضع خطوطه العريضة و الإلمام بكل جوانبه قبل البداية في برمجته و تدوين كل ذلك. فأغلب المبرمجين يهملون ما أشير إليه, يتركون هته الأفكار في رؤوسهم حتى إذا كثرت اختلطت عليهم.

- كن على اطلاع بآخر المستجدات المتعلقة باللغة التي تبرمجعليهاو الأدوات التي تعمل بها و حاول مواكبة تطورها ليكون برنامجك مجهزا بأحدث التقنيات.

- ابتعد عن التلفاز الفيسبوك! فهما من أكبر المضيعات للوقت من جهة و من جهة أخرى ففيهما فيروس فتاك, فتاك بما يحشو في العقل من أخطاء و أحلام يقظة و ابعادك عن هدفك و طموحك. ألم تسمع قول من قال بشأن مستخدم التقنية : إن كان ما تستخدمه مجانا فاعلم أنك أنت السلعة.

-ما يفيد كذلك : التجول في الطبيعة من وقت لآخر بعيدا عن التقنية و قريبا من الهواء الطلق و مناظر الطبيعة المريحة و ذلك من أجل تصفية المزاج و تنقية الأفكار و إراحة الجسم.

- أتقنت ُالبرمجة ؟ ثم ماذا؟ حتما ستصل إلى هذا المنعطف أو بالأصح مفترق الطرق : إما أن تستمر في البرمجة و تؤسس شركتك وتطرح منتوجات منافسة و تعمل على أن يكون لك كيان أمام الآخرين ممن سبقوك و إما أن تطرح سيرتك الذاتية في إحدى الشركات ممن يعطي لجهدك و موهبتك التي قمت بصقلها حقها لتلتحق للعمل لديهم, و إما أن تضع حدا لهذه الأفكار الهلوسة ثم تتنهد و تقول : لا بأس تجربة جميلة, و ذلك دون أن تلاحظ أن عشر سنوات قد مرت و أنت جالس أمام شاشة الحاسوب. فالمسألة ليست مجرد كتابة للأكواد البرمجية بل تتخطى كل ذلك فقد تصل إلى أن تصبح مصدر رزقك أو ثروتك, لذلك لما أخبرتك أن تترك الملهيات و تتمسك بالمجديات فالأمر في غاية الأهمية.

- الفكرة: نقطة البداية و الهدف في النهاية. إذا لم تكن لديك فكرة واضحة كل الوضوح, فدعني أخبرك أنك كمن تعلم السباحة في بركة ثم وجد نفسه في بحر, نعلم انه لن يغرق لكن إلى أين يا هذا؟! ماذا تريد أن تحقق ؟ أما إن كنت ممن يعي ما يصنع فالطريق من هنا : الفكرة إما مُنتَج يعالِج مشكل معين أو خدمة يحتاجها الناس أو تسهل لهم مهمة معينة. فإذا قدمت منتوجك للناس و جربوه و قبلوا به فقد تصبح عليك مسؤوليات جديدة و كبيرة : تحديثات دورية, إضافة خصائص جديدة, معالجة مشاكل قديمة, معرقلات إدارية و سياسية و جغرافية, فأنت الآن أصبحت المبرج الماهر الذي يدير شركة ناشئة و التي تقدم منتوجات أو خدمات لكل أصناف و أجناس و أنواع الناس من كل صوب و في كل بقاع العالم.

-ضع نفسك مكان الزبون. اتخذ لنفسك ثلة ممن تثق فيهم يجربون منتوجك و يدفعونه لأبعد حدوده ثم يدلون عليك بما رءوا فيه من عيوب أو نقائص. بل تحدث مع زبائنك لتكتشف الأفكار التي قد تضيفها إلى برنامجك ليزيد من رضاهم به.

- اعتبر من قصص الكبار ممن فاتك. فشركة أبل تأسست في مرآب و شركة سناب شات تأسست على يد شاب في العشرينيات و فيسبوك بدأ كموقع بسيط لا يعرف حتى مؤسسه المدى الذي سيصل إليه. لذلك لا تستهن أبدا بنفسك و لتكن ثقتك بنفسك من حديد.

- في النهاية, تذكر ان زكاة العلم نشره و أنه لن يُنقِص من علمك شيئا علمته لغيرة, بل العكس تماما التعليم كالمراجعة, فهو تمرين و ترويض للذاكرة و العقل و يجعلك على دراية دائمة بما تعلم. و من يعلم فقد تُعلم من سيصبحون طاقم عملك و محرك شركتك مستقبلا.

و الله الموفق للخير و ما فيه الصلاح للبلاد و العباد.

دائما فكر من جديد - بقلم delphi4us

عند تصميم قواعد البيانات او اي برنامج او مكون او كلاس ، يجب التفكير في ما تفعل مرة واكثر من مرة .

ذات مرة كنت بصدد تصميم قاعدة بيانات لبرنامج الاكواد، وطبعاً كان التحليل كالتالي :

جدول الاكواد	جدول المستندات الفرعية	جدول للمجلدات الرئيسية
رقم التعريف التلقائي	رقم التعريف التلقائي	رقم التعريف التلقائي
حقل لاسم الكود	حقل لاسم المستندات	حقل لاسم المجلدات
حقل الفهرسة لجدول المستندات	حقل الفهرسة لجدول المستندات	

بحيث تكون النتيجة كالتالي

طبعاً نجحت التجربة ولكن فيها بعض الاشياء التي ضلت مقلقة وتشعر ان هناك شئ غير منطقي بالتصميم .
وظللت احاول تطوير البرنامج اكثر من مرة
حتى توصلت الى الطريقة الصحيحة
والعيب كان في تصميم قاعدة البيانات نفسها
اذا كيف كانت يجب ان تكون ...؟
بداية دعونا نشاهد التصميم السابق
من تتبعة نجد ان هناك تكرار غير مطلوب
ما هو؟



رقم التعريف مكرر ثلاث مرات، وحقل الاسم مكرر ثلاث مرات.
اذا لماذا ثلاث جداول بدل جدول واحد ، دعونا نشاهد النتيجة.

جدول عام	لو كان حقل الفهرسة يساوي صفر فاحتواه يعني انه مجلد ولو كان يحتوي رقم فهو رقم التعريف لمجلد أو مستند وبذلك فكل ما عليك البحث عن رقم التعريف لتعرف لمن يعود بالضبط وتملاء وتتعامل مع البيانات على أساس ذلك وبذلك وكما ترون تم اختصار الكثير بعد إعادة التفكير
رقم التعريف التلقائي	
حقل الاسم	
حقل الفهرسة	